ORIGINAL RESEARCH

A enhanced algorithm for floorplan design using evolutionary technique

Dhiraj, Seema Verma, Rajesh Kumar, Himanshu Choudhary

Digital Systems Group, Council of Scientific & Industrial Research-(CSIR-CEERI), Pilani, Rajasthan, India.

Correspondence: Dhiraj. Address: Digital Systems Group, Council of Scientific & Industrial Research-(CSIR-CEERI), Pilani, Rajasthan, India. Telephone: 91-159-625-2270. Email: dhiraj@ceeri.ernet.in.

Received: June 27, 2012 **Accepted:** August 14, 2012 **Published:** December 1, 2012
DOI: 10.5430/air.v1n2p38 **URL:** http://dx.doi.org/10.5430/air.v1n2p38

Abstract

Floor planning is an important problem in very large scale integrated-circuit (VLSI) design automation domain as it evaluates the performance, size, yield and reliability of ICs. Due to rapid increase in number of components on a chip, floor planning has gained its importance further in determining the quality of the design achieved. In this paper we have devised an approach for placement of modules in a given area with bounding constraints in terms of minimum placement area imposed. We have used Modified Genetic Algorithm (MGA) technique for determining and obtaining an optimal placement using an iterative approach.

Key words

Interconnect, Genetic algorithm, Bounding rectangle, Wire length. Modified genetic algorithm (MGA)

1 Introduction

1.1 VLSI issues

Due to the rapid increase in complexity of chips, circuit sizes are getting larger. Various functional modules called as IP blocks are intensively used to reduce the design complexity. It makes floor planning a critical process. Major problem in this lies in the geometrical relation between modules. The representation directly affects the quality of floor plan /placement and the complexity of placing them to their required coordinates.

1.2 Floor planning as an optimization problem

The task of VLSI physical design is to produce the layout of an integrated circuit. Genetic Algorithm is a tool of the artificial intelligence community [1]. It is a paradigm for examining a state space which produces its solution through simultaneous consideration and manipulation of a set of possible member solutions. The output of this step is the layout of a physical package that optimally or near optimally realizes the logical representation. The objectives to this problem are minimizing area and reduce wire length for nets, maximize rout ability and determine shapes of flexible blocks.

Rectangle packing problem: RP

Let M be a set of m rectangular modules of fixed orientations, whose heights and widths are given in real numbers. The decision of our problem RP (A) is to decide whether M can be packed onto a chip of area A [2]. It can be shown that RP(H,W) can be polynomially reducible to an instance of RP (A) by formulas, where different parameters are redefined using the conversion given below by equations (1) to (3):

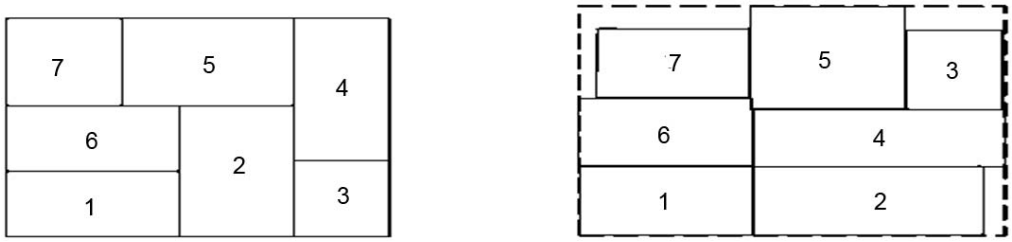
$$r \leftarrow \frac{the\ maximum\ width\ over\ modules}{2H} \tag{1}$$

$$A \leftarrow (W + rH)(W + 2rH) \tag{2}$$

$$M' \leftarrow \{(\ w\ X\ rh\ )|\forall\ (w\ X\ h) \in M\} \cup \{\ rH\ X\ rH, (\ W + rH)X\ (W + rH)\} \tag{3}$$

Where M is a set of m rectangular modules of fixed orientations whose heights (h) and widths (w) are given in real numbers r, A is the area in which M needs to be packed. W is the width and H is the height of RP i.e. Rectangular Packed Modules.

The figure 1 depicts the importance and effectiveness of floorplan in terms of Area acquired.



Optimal floorplan of soft modules in terms of Area Non Optimal floorplan of soft modules

Figure 1. Effect of floorplanning on circuit area utilized

The paper focuses on placement problem out of partitioning, placement and routing steps in layout problem. It includes the assignment of circuit elements to locations on the chip. The objective of placement is to minimize the layout area of the chip [3-5].

1.3 Techniques of floor planning

There are various techniques of representing floorplans but mainly divided in two categories. One is slicing structures in which a binary tree is used. Wong et al. had proposed a technique that traverses the binary tree in a postorder called polish expression to represent slicing floorplan. It has some benefits such as smaller encoding cost and faster runtime. In real designs optimal solutions might not be in the solution space of slicing structure. The other structure of representation is nonslicing which includes Sequence Pair (SP), O-tree, B-tree, Corner Block List (CBL), and Transitive Closure Graph (TCG). These nonslicing representations need more evaluating runtime than the polish expression [6].

Our Algorithm is based on Slicing floor plan which is a rectangular floor plan with n basic rectangles that can be obtained by recursively cutting a rectangle into smaller rectangles using a series of vertical and horizontal cuts. It can be represented in the form of a binary tree called a slicing tree in which each internal node of the tree is labeled either '*' or '+' corresponding to vertical or horizontal cut respectively. Each slicing tree can be represented alternatively using a postfix expression. This representation can only present slicing structure of a floorplan. Each packing is represented in an encoded sequence, including module name and two relational operators. We can obtain a polish expression of length 2n – 1 with n

modules in the slicing floorplan by traversing the slicing tree. The postfix expression is derived by carrying out a post order traversal [7, 8].

The approach adopted in our algorithm for obtaining the placement of the circuit Modules is shown as figure 2.

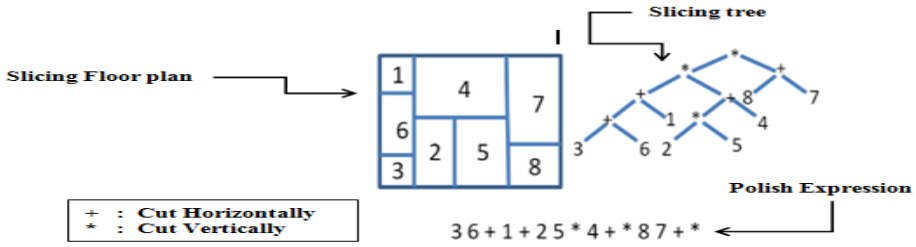


Figure 2. Slicing Floor plan and Postfix Expression

1.4 Genetic algorithm: A Brief description

The genetic algorithm is an optimization and search technique based on the principles of natural selection. It allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e. minimizes the cost function).

It starts with a set of randomly generated possible solutions called as chromosomes. During every iteration, the population elements are evaluated according to their fitness which in placement is interpreted in terms of Bounding Rectangle for generating offspring's through crossover, mutation and inversion technique. Due to stochastic process, the fitter parents stand good chance of producing better placements. It leads to improvement in overall population. The success of genetic algorithm depends on its choice of various parameters and functions which control its execution like selection principle, crossover probability and technique adopted mutation [9, 10]. The general flow of GA is shown in figure 3.

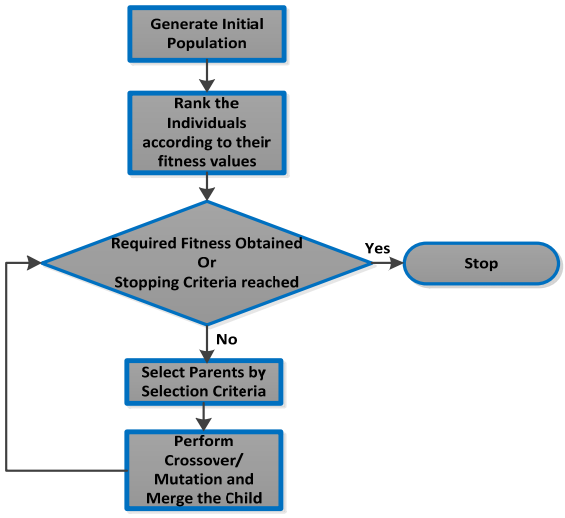


Figure 3. Genetic Algorithm Flow Steps

1.5 Organization of paper

The paper is organized as follows. Section 2 describes the problem formation using Evolutionary technique i.e. MGA. Implementation of the proposed technique is given in section 3. Section 4 gives the experimental results on two test cases.

Section 5 gives the detailed outline regarding the results and discussions. Section 6 is the final outcome of the work in terms of conclusion.

2 Problem description

The input to our placement problem is a set of N circuit blocks or Modules M = {m_1,m_2 …..m_m} and a set of signals called as Nets N = {n_1,n_2…n_n}. The task is to assign each module to its respective position with subject to constraint that the routing space required should be minimum or the bounding rectangle which encapsulates all the placed blocks is having a minimum effective area [11]. There are two kinds of blocks in floorplanning:

1) Hard Block: Its shape is fixed, and is denoted as (W, H), where W is the width and H is the height of module.
2) Soft Block: Its area is also fixed, but the ratio of width/height is included in a given range. It has a range of aspect ratio also specified in addition to width and height. The range specifies the minimum and maximum aspect ratios permitted for placement of given block.

The placement is an important step in VLSI design as an inferior placement will not only affect the chips performance but might also make it non realizable by producing excessive wire length which is beyond the available routing resources. So, a placement problem being an optimization problem reduces the chip size and makes it faster as timing constraints are fulfilled. In this paper we have focused on reducing the total dead space of the chip by placing the blocks either adjacent or above each other such that the final dead space of the circuit will be minimum [12, 13].

For a VLSI floorplanning of n modules, the cost may be defined in equation (4) as follows:

$$Cost(F) = W_1 \frac{Area(F)}{Area*} + W_2 \frac{Wirelength\ (F)}{Wirelength*} \quad (4)$$

Where $Area(F)$ is the area cost which is the measured by the area of the smallest rectangle enclosing all the modules, $Wirelength\ (F)$ is the interconnection cost which is the total length of the wires fulfilling the interconnections specified by net list N. Area* and Wire length * represent the minimal area and interconnection costs respectively, W_1 and W_2 are weights assigned to the area minimization objective and the interconnection minimization objective, respectively where

$$0 \leq W_1, W_2 \leq 1, and\ W_1 + W_2 = 1$$

Minimizing the overall placement area is also related to minimizing the routing space or reducing the signal propagation delay. It is a case dependent process as by putting a restriction on wiring length may provide a better protection to timing restriction or the restriction on placement area of cells may be required to maintain the circuit character [14, 15].

3 Problem implementation: Genetic formulation

The GA is a class of heuristic algorithm which learns as it continues its search for better placemen during its evolution.

The initial population will continue to evolve by discarding the bad solutions with the more fit arrangements.

The random generation of modules is biased such that it leads to reduced dead space so as to improve the scores of initial population. The VLSI floorplan is a minimization problem in which the objective is to minimize the cost of floorplan F. The fitness of an individual is defines in equation (5) as follows:

$$f(m) = \frac{1}{Cost(f(m))} \quad (5)$$

Where $f(m)$ is corresponding floor plan of m i.e. number of blocks.

The fitness based selection is used to choose the parents for crossover, two parents one acting as a target parent while other acting as a passing parent where the passing parent is used to redefine the arrangement in the target parent. The mutation operator uses a directed evolution approach and tries to reduce the bounding rectangle of arrangement [16, 17].

The effectiveness of a GA is defined by its randomness and the stopping criteria adopted. Our algorithm will stop when it reaches the steady state or if no improvement results in successive number of generation after trying the different arrangement possibilities. Since GA is sort of random approach the results can be different from one run to another. So we have adopted an average feature of some N iteration [18].

4 Circuit evaluation

4.1 Algorithm implementation

The algorithm starts with the formation of initial population. There are various techniques for this but usually it is a bitmapped encoded solution. We have used postfix expression for depicting the population members.

Using series maker function with emphasis on reduction in dead space. It then repetitively runs the crossover, selection with merge and mutation operations in the main loop. The figure 4 shows the flow of the algorithm used.

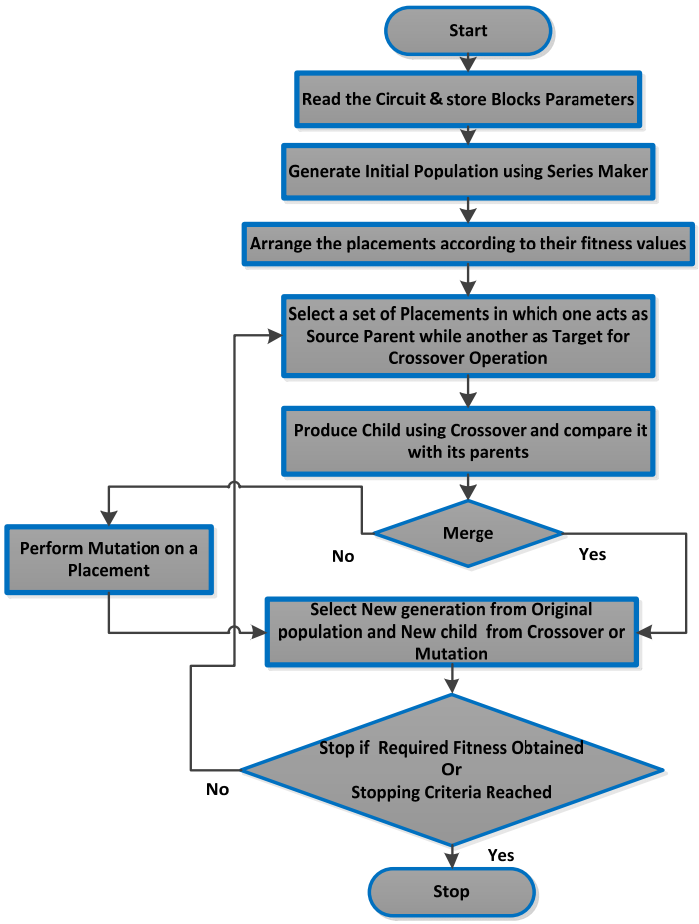


Figure 4. Flowchart showing the Algorithm Implementation

 ISSN 1927-6974 E-ISSN 1927-6982

4.2 Case study

In order to verify the functionality of the approach we have tested it on two different test cases as shown in Table 1.

Case Study I – Test Case I

The first Case is a circuit with 15 Modules. Each Module is specified with its number and also the dimensions. The modules considered are all rectangular blocks with varying dimensions. The maximum length is 15 and maximum breadth is also 17 for this case.

Case Study II – Test Case II

The second case study is another circuit but with 25 blocks. The number of modules considered for a given placement indirectly relates with the complexity with which it can be placed without leveraging the placement constraints. The maximum length is 16 and maximum breadth is 17.

Table 1. Test Case Study

Test Case I (15 Blocks)		**Test Case II (25 Blocks)**	
	15		25
1	6 8	1	8 6
2	10 13	2	13 10
3	12 10	3	10 12
4	5 10	4	10 5
5	5 12	5	12 4
6	6 10	6	10 6
7	15 10	7	7 15
8	11 8	8	8 15
9	9 7	9	7 9
10	10 7	10	7 10
11	7 5	11	5 7
12	10 13	12	13 10
13	8 15	13	15 8
14	5 14	14	14 5
15	8 17	15	8 17
		16	16 8
		17	9 13
		18	16 10
		19	10 14
		20	15 10
		21	14 12
		22	10 7
		23	11 8
		24	9 17
		25	10 7

4.3 Module description

The functionality of different blocks are hereby explained.

- **Series Maker:** It generates the first placement of blocks i.e. population P1. It accepts the various details from circuit file regarding its dimensions and arranges them in a pattern by using horizontal i.e. 43 or vertical operator i.e. 42. The arrangement of blocks is done purely on random basis so as to introduce a variety in the population.

Two operators (H or V) operate on operands (blocks bi) which are circuit modules by arranging the blocks either horizontally or vertically. The floor plan returned by the function is LB (Left and Bottom compact) complete type.

- **Cost Calculator:** It calculates the fitness of the population members in terms of the Bounding Rectangle area. It accepts the dimension of the members from the structural elements and performs an analysis in terms of the Effective Area and Dead space in each arrangement. If two blocks are related together with relational operator '*', it means first block is above the second and with relational operator '+' means first block is in right of second. So accordingly it calculated the effective area of the combined arrangement of blocks.

- **Minimum Effective Area:** It sorts the population elements according to their effective area and finds the placement which gives the minimum Bounding Rectangle area.

- **Merge:** It is called after the Crossover operation or Mutation operations. Its job is to calculate whether the generated offspring is having fitness more than the Parent members or not. If it founds a better child then it replaces the respective parent element with that child. Since two children are generated after crossover so it makes a decision in terms of both. It no better child generation takes place its operation will get skipped and no merge will take place.

  Pseudo code for Merge after Crossover

  1) Compare the fitness value of Children C1 and C2 with their parents considered for Crossover i.e. P1 & P2.
  2) Replace the respective P1 or P2 if it has lesser fitness value then the corresponding child.
  3) If no replacement possible ignore the child.

  Pseudo code for Merge after Mutation

  1) Compare the fitness value of blocks generated after Mut1, Mut2 & Mut3 with each other.
  2) Select the mutation giving maximum improvement.
  3) Replace the Reference placement with the mutation of step 2.

- **Crossover:** This function uses probability based parent selection out of population elements and chooses two placements as parents. One acts as passing parent while other acts like a target parent. The block members from passing parent are arranged according to the arrangement in the target parent. With this operation since both parents swap their roles we have two children generated from this operation (see Figure 9 -11 in Test case I and figure 21-23 in Test case 2).

  Pseudo code for Crossover

  1) Arrange Population Pi according to probability of selection i.e. [Pi] = b1, b2, b3…….bm.
  2) Select Parent1 & Parent2 where Parent1 is Passing parent and Parent2 is target parent C1 = [P1], [P2].
  3) Block placements from P1 are stored in a structure S1 = b1, b2……bm and Block Arrangements are stored in Structure S2 = 42, 43, 43…..42.
  4) Children C1 and C2 are created by using the Blocks from S1 and their relative Placements from S2 and vice Versa in Child C2.

 ISSN 1927-6974 E-ISSN 1927-6982

- **Mutation:** Mutation operation is performed to introduce some new feature in the placement which is not possible by simply exchange of arrangements in crossover. So we have used three variants of Mutation i.e. Mut1, Mut2 & Mut3. In Mut1 two randomly chosen block members exchange their positions. In Mut2 relation between any two block are reversed i.e. their positional relation gets interchanged the horizontally placed elements becomes vertically placed and vice versa. In Mut3 a finite length of the encoded string is taken and its reciprocal is considered for placement.

  Refer to figures 12-13 for mutation in Test case I and figures 24-25 for mutation in Test case 2.

  Pseudo code for Mutation

  1) Consider a Placement as Pi = b1, b2, 42, b3, 43, 42…43.
  2) In Mut1: Randomly select two blocks in this Placement and swap their positions
  3) In Mut2: Randomly select two placement operators i.e 42, 43 and swap them
  4) In Mut3: Randomly select two points choose a finite length string between them and invert the blocks with operators in that length.
  5) Out of different variants of Mutation one which gives maximum improvement in fitness value is considered While rest are discarded

- **Draw:** This function has the role of producing the visualization of the operations performed in the main loop. It produces an arrangement of blocks in a placement according to their relative placements along the axes.

- **Mark:** This function is used to produce a randomly generated relation between blocks. Two relations can be possible between neighboring blocks either they are horizontally aligned with each other i.e. '+'. Or they are vertically aligned with each other i.e. '*'.

Table 2. Effect of Crossover and Mutation

Iteration Number	Population Elements	Effect of Crossover	Effect of Mutation
1.	Sol1:67*99,Sol2:103*43	Child1:69*81,Child2:99*46	Not Performed
2.	Sol1:99*46,Sol2:103*43	Child1:103*43,Child2:99*46	Not Performed
3.	Sol1:103*43,Sol2:103*43	Child1: 103*43,Child2: 103*43	Child1:103*43,Child 2: 103*43 No Replacement
4.	Sol1:103*43,Sol2:103*43	Child1: 103*43,Child2: 103*43	Child1:113*30,Chid2:103*43 Replacement of sol1 by child1
5.	Sol1:113*30,Sol2:103*43	Child1:101*40,Child2:113*30	Not Performed
6.	Sol1:113*30,Sol2:113*30	Child1: 113*30,Child2: 113*30	Child1:113*30,Child2:113*30 No Replacement
7.	Sol1:113*30,Sol2:113*30	Child1: 113*30,Child2: 113*30	Child1:119*22,Chid2:119*22 Replacement of sol1 by child1 Replacement of sol2 by child2
8.	Sol1:119*22,Sol2:119*22	Child1:119*22,Child2:119*22	Child1: 119*22,Child 2: 119*22 No Replacement
9.	Sol1:119*22,Sol2:119*22	Child1: 119*22,Child2: 119*22	Child1: 119*22,Child 2: 119*22 No Replacement
10.	Sol1:119*22,Sol2:119*22	Child1: 119*22,Child2: 119*22	Child1: 119*22,Child 2: 119*22 No Replacement

The effect of various convergence operators used in GA in optimizing towards a Global optimal placement are shown in Table 2 and the simulation results in terms of Bounding Rectangle and Dead Space are shown in Table 3. The effective Area utilization has increased up to 48 % in just 10 iterations and the contribution of Dead Space reduced up to 51 %.

Table 3. Simulation Results iteration wise showing reduction in terms of dead space Operator

Iteration Number	Population Elements	Bounding Rectangle Area	Dead Space	Effective Area Utilization (%)	Dead Space Contribution (%)	Reduction in Dead Space
1.	Sol1:67*99	Sol1:6633	Sol1:5360	19.19192	80.80808081	2079
	Sol2:103*43	Sol2:4429	Sol2:3156	28.74238	71.25762023	0
2.	Sol1:99*46	Sol1:4554	Sol1:3281	27.95345	72.04655248	125
	Sol2:103*43	Sol2:4429	Sol2:3156	28.74238	71.25762023	0
3.	Sol1:103*43	Sol1:4429	Sol1:3156	28.74238	71.25762023	0
	Sol2:103*43	Sol2:4429	Sol2:3156	28.74238	71.25762023	0
4.	Sol1:103*43	Sol1:4429	Sol1:3156	28.74238	71.25762023	1039
	Sol2:103*43	Sol2:4429	Sol2:3156	28.74238	71.25762023	0
5.	Sol1:113*30	Sol1:3390	Sol1:2117	37.55162	62.44837758	0
	Sol2:103*43	Sol2:4429	Sol2:3156	37.55162	71.25762023	1039
6.	Sol1:113*30	Sol1:3390	Sol1:2117	37.55162	62.44837758	0
	Sol2:113*30	Sol2:3390	Sol2:2117	37.55162	62.44837758	0
7.	Sol1:113*30	Sol1:3390	Sol1:2117	37.55162	62.44837758	772
	Sol2:113*30	Sol2:3390	Sol2:2117	37.55162	62.44837758	772
8.	Sol1:119*22	Sol1:2618	Sol1:1345	48.6249	51.37509549	0
	Sol2:119*22	Sol2:2618	Sol2:1345	48.6249	51.37509549	0
9.	Sol1:119*22	Sol1:2618	Sol1:1345	48.6249	51.37509549	0
	Sol2:119*22	Sol2:2618	Sol2:1345	48.6249	51.37509549	0
10.	Sol1:119*22	Sol1:2618	Sol1:1345	48.6249	51.37509549	0
	Sol2:119*22	Sol2:2618	Sol2:1345	48.6249	51.37509549	0

5 Results and discussion

5.1 Test case circuit I: Results

The proposed technique is successfully tested over several test cases; the results of two among them are shown here.

The first test case is having 15 blocks randomly chosen with maximum length as 15 and maximum breadth as 17.

The initial solutions i.e. Solution 1 and Solution 2 constituting first population have following properties:

> Solution 1: Breadth: 99, Length: 67, Effective area: 6633, Dead space: 5360. Relative Placement of Blocks: [5,14,43,11,10,1,4,42,6,43,42,42,42,8,43,13,43,15,43,2,43,3,43, 9, 42, 7, 12, 42, 42, 42]

> Solution 2: Breadth: 43, Length: 103, Effective area: 4429, Dead space: 3156. Relative Placement of Blocks: [10,15,7,42,12,42,42,9,42,8,42,14,1,43,13,4,11,3,2,6,42,5,42, 43, 42, 43, 42, 42, 42, 42]

Here in relative placement 42 shows the vertical arrangement of blocks i.e. + operator (cut horizontally) and 43 shows the horizontal arrangement of blocks i.e. * operator (cut vertically) of figure 2.

The effective area is the area of the bounding Rectangle which has all modules placed to their respective positions. Dead space is the unutilized space of the bounding rectangle. So our target is to rearrange the blocks so that the dead space decreases and effective area also reduces respectively.

 ISSN 1927-6974 E-ISSN 1927-6982

After running over 10 iterations the Final Placement as shown in Table 3 is:

Final Solution: Breadth: 22, Length: 119, Effective area: 2618, Dead space: 1345

Relative Placement of Blocks: [10, 7, 15, 42, 12, 42, 42, 9, 42, 8, 42, 14, 1, 42, 13, 4, 11, 2, 3, 6, 42, 5, 42, 42, 42, 43, 42, 42, 42]

So, the Effective area got reduced from 6633 to 2618 and Dead space 5360 to 1345 in 10 iterations. The Combined Blocks area is 1273.

The Crossover operation at its best replaces a parent with 67*99 dimensions and effective area as 6633 by a child having 99×46 dimensions and effective area as 4554 as shown in Table 2.The Mutation operator at its best replaces a block with 113×30 dimensions with dead space 2117 by block 119×22 dimensions with dead space of 1345 as shown in Table 2.

It has been found experimentally that final solution will have more compactly packed blocks if we have more number of initial solutions. By using 3 initial solutions instead of 2 we can reduce the final placement area to 2160 from present 2618 as shown in figures 5 to 8. Figures 9-17 shows the results of crossover, mutation and variations of effective area in different solutions respectively.

With further increase in this pattern we are able to reduce effective area to 1989 with 10 initial solutions but the final price may be obtained for an increased number of iterations since GA is random search approach so it reaches a global minimum finally. The Simulation time on Core2Duo @3.00 GHZ machine with 1 GB RAM using MATLAB 7 environment is 0.929000 seconds.

5.2 Test case circuit II: Results

The second test case is having 25 blocks randomly chosen with maximum length as 16 and maximum breadth as 17.

Table 5 shows the net reduction in Dead space from 17900 to 2753 with effective reduction of placement area from 20449 to 5302. The combined Blocks Area is 2549.

The initial solutions i.e. Solution 1 and Solution 2 (see Figure 18 & 19) constituting first population have following properties:

1) Solution 1: Breadth: 121, Length: 169, Effective area: 20449, Dead space: 17900. Relative Placement of Blocks: [1, 24, 43, 5, 42, 23, 42, 25, 42, 10, 43, 2, 43, 17, 6, 9, 15, 43, 4, 16, 20, 42, 42, 8, 43, 3, 7, 18, 43, 22, 42, 14, 42, 42, 42, 13, 42, 19, 42, 21, 11, 43, 43, 12, 43, 43, 42, 42, 42, 42]

2) Solution 2: Breadth: 108, Length: 145, Effective area: 15660, Dead space: 13111. Relative Placement of Blocks: [1, 9, 43, 3, 22, 43, 10, 5, 43, 17, 2, 42, 18, 42, 13, 43, 42, 42, 42, 12, 6, 21, 4, 43, 14, 11, 24, 8, 15, 25, 43, 16, 20, 19, 43, 7, 42, 42, 42, 42, 23, 43, 43, 43, 43, 43, 42, 42, 42, 42]

After running over 27 iterations the Final Placement as shown in Table 5 and figure 20 is

Final Solution: Breadth: 22, Length: 241, Effective area: 5302, Dead space: 2753

Relative Placement of Blocks: [1, 9, 42, 3, 22, 42, 10, 5, 43, 18, 17, 42, 2, 42, 13, 42, 42, 42, 42, 6, 12, 21, 4, 42, 14, 11, 24, 8, 16, 15, 42, 25, 20, 19, 42, 7, 42, 42, 42, 42, 23, 42, 42, 42, 42, 43, 42, 42, 42]

In relative placement 42 shows the vertical arrangement of blocks and 43 shows the horizontal arrangement of blocks.

The Crossover operation at its best replaces a parent with 169*121 dimensions and effective area as 20449 by a child having 139*123 dimensions and effective area as 17097as shown in Table 4 and figures 21 to 23. The Mutation operator at its best replaces a block with 148*102 dimensions by block 163*92 dimensions as shown in Table 4 and figures 24 to 25.

Table 4. Effect of Crossover and Mutation Operator

Iteration Number	Population Elements	Effect of Crossover	Effect of Mutation
1.	Sol1:169*121,Sol2:145*108	Child1:170*106,Child2:139*123	Not Performed
2.	Sol1:139*123,Sol2:145*108	Child1:145*108,Child2:139*123	Not Performed
3.	Sol1:145*108,Sol2:145*108	Child1: 145*108,Child2: 145*108	Child1:145*108,Child 2: 148*102 Replacement of sol1 by child2
4.	Sol1:148*102,Sol2:145*108	Child1:145*108,Child2: 148*102	Not Performed
5.	Sol1:148*102,Sol2:148*102	Child1:148*102,Child2:148*102	Child1:158*95,Child 2: 163*92 Replacement of sol1 by child1 Replacement of sol2 by child2
6.	Sol1:158*95,Sol2:163*92	Child1: 158*95,Child2: 163*92	Not Performed
7.	Sol1:163*92,Sol2:163*92	Child1:163*92,Child2: 163*92	Child1:163*92,Chid2:172*75 Replacement of sol1 by child2
8.	Sol1:172*75,Sol2:163*92	Child1:163*92,Child2:172*75	Not Performed
9.	Sol1:172*75,Sol2:172*75	Child1:172*75,Child2: 172*75	Child1: 172*75,Child 2: 177*68 Replacement of sol1 by child2
10.	Sol1:177*68,Sol2:172*75	Child1:172*75,Child2: 177*68	Not Performed
11.	Sol1:177*68,Sol2:177*68	Child1:177*68,Child2:177*68	Child1: 188*60,Child 2: 192*60 Replacement of sol1 by child1 Replacement of sol2 by child2
12.	Sol1:188*60,Sol2:192*60	Child1:192*60,Child2:188*60	
13.	Sol1:188*60,Sol2:188*60	Child1:188*60,Child2: 188*60	Child1: 198*54,Child 2: 198*55 Replacement of sol1 by child1 Replacement of sol2 by child2
14.	Sol1:198*54,Sol2:198*55	Child1:198*55,Child2:198*53	
15.	Sol1:198*54,Sol2:198*53	Child1:198*53,Child2:198*54	
16.	Sol1:198*53,Sol2:198*53	Child1:198*53,Child2:198*53	Child1: 208*48,Child 2: 198*53 Replacement of sol1 by child1
17.	Sol1:208*48,Sol2:198*53	Child1:198*53,Child2:208*48	
18.	Sol1:208*48,Sol2:198*53	Child1:208*48,Child2:208*48	Child1: 223*41,Child 2: 212*48 Replacement of sol1 by child1
19.	Sol1:208*48,Sol2:208*48	Child1:210*50,Child2:223*41	
20.	Sol1:223*41,Sol2:223*41	Child1:223*41,Child2:223*41	Child1: 223*41,Child 2: 233*37 Replacement of sol1 by child2
21.	Sol1:233*37,Sol2:223*41	Child1:217*42,Child2:233*37	
22.	Sol1:233*37,Sol2:233*37	Child1:233*37,Child2:233*37	Child1: 233*37,Child 2: 233*37 No Replacement
23	Sol1:233*37,Sol2:233*37	Child1:233*37,Child2:233*37	Child1: 241*22,Child 2: 233*37 Replacement of sol1 by child1
24.	Sol1:241*22,Sol2:233*37	Child1:233*37,Child2:241*22	
25.	Sol1:241*22,Sol2:241*22	Child1:241*22,Child2:241*22	Child1: 246*24,Child 2: 241*22 No Replacement
26.	Sol1:241*22,Sol2:241*22	Child1:241*22,Child2:241*22	Child1: 231*25,Child 2: 253*22 No Replacement
27.	Sol1:241*22,Sol2:241*22	Child1:241*22,Child2:241*22	Child1: 241*22,Child 2: 241*22 No Replacement

ISSN 1927-6974 E-ISSN 1927-6982

The unused Placement area in terms of Dead space is reduced from 17900 to 2753 and Bounding Rectangle Area from 20449 to 5302 as shown in Table 5.

Table 5. Simulation Results iteration wise showing reduction in terms of dead space

Iteration Number	Population Elements	Bounding Rectangle Area	Dead Space	Effective Area Utilization (%)	Dead Space Contribution (%)	Reduction in Dead Space
1.	Sol1:169*121 Sol2:145*108	Sol1:20449 Sol2:15660	Sol1:17900 Sol2:13111	12.46515722 16.27713921	87.53484278 83.72286079	3352 0
2.	Sol1:139*123 Sol2:145*108	Sol1:17097 Sol2:15660	Sol1:14548 Sol2:13111	14.90904837 16.27713921	85.09095163 83.72286079	1437 0
3.	Sol1:145*108 Sol2:145*108	Sol1:15660 Sol2:15660	Sol1:13111 Sol2:13111	16.27713921 16.27713921	83.72286079 83.72286079	564 0
4.	Sol1:148*102 Sol2:145*108	Sol1:15096 Sol2:15660	Sol1:12547 Sol2:13111	16.88526762 16.27713921	83.11473238 83.72286079	90 564
5.	Sol1:148*102 Sol2:148*102	Sol1:15096 Sol2:15096	Sol1:12547 Sol2:12547	16.88526762 16.88526762	83.11473238 83.11473238	86 100
6.	Sol1:158*95 Sol2:163*92	Sol1:15010 Sol2:14996	Sol1:12461 Sol2:12447	16.98201199 16.9978661	83.01798801 83.0021339	14 0
7.	Sol1:163*92 Sol2:163*92	Sol1:14996 Sol2:14996	Sol1:12447 Sol2:12447	16.9978661 16.9978661	83.0021339 83.0021339	2096 0
8.	Sol1:172*75 Sol2:163*92	Sol1:12900 Sol2:14996	Sol1:10351 Sol2:12447	19.75968992 16.9978661	80.24031008 83.0021339	0 2096
9.	Sol1:172*75 Sol2:172*75	Sol1:12900 Sol2:12900	Sol1:10351 Sol2:10351	19.75968992 19.75968992	80.24031008 80.24031008	864 0
10.	Sol1:177*68 Sol2:172*75	Sol1:12036 Sol2:12900	Sol1:9487 Sol2:10351	21.17813227 19.75968992	78.82186773 80.24031008	0 864
11.	Sol1:177*68 Sol2:177*68	Sol1:12039 Sol2:12036	Sol1:9487 Sol2:9487	21.1977739 21.17813227	78.8022261 78.82186773	756 516
12.	Sol1:188*60 Sol2:192*60	Sol1:11280 Sol2:11520	Sol1:8731 Sol2:8971	22.59751773 22.12673611	77.40248227 77.87326389	0 240
13.	Sol1:188*60 Sol2:188*60	Sol1:11280 Sol2:11280	Sol1:8731 Sol2:8731	22.59751773 22.59751773	77.40248227 77.40248227	588 390
14.	Sol1:198*54 Sol2:198*55	Sol1:10692 Sol2:10890	Sol1:8143 Sol2:8341	23.8402544 23.40679522	76.1597456 76.59320478	0 396
15.	Sol1:198*54 Sol2:198*53	Sol1:10692 Sol2:10494	Sol1:8143 Sol2:7945	23.8402544 24.29007052	76.1597456 75.70992948	198 0
16.	Sol1:198*53 Sol2:198*53	Sol1:10494 Sol2:10494	Sol1:7945 Sol2:7945	24.29007052 24.29007052	75.70992948 75.70992948	510 0
17.	Sol1:208*48 Sol2:198*53	Sol1:9984 Sol2:10494	Sol1:7435 Sol2:7945	25.53084936 24.29007052	74.46915064 75.70992948	0 510
18.	Sol1:208*48 Sol2:198*53	Sol1:9984 Sol2:9984	Sol1:7435 Sol2:7435	25.53084936 25.53084936	74.46915064 74.46915064	841 0
19.	Sol1:208*48 Sol2:208*48	Sol1:9143 Sol2:9984	Sol1:6594 Sol2:7435	27.87925189 25.53084936	72.12074811 74.46915064	0 841
20.	Sol1:223*41 Sol2:223*41	Sol1:9143 Sol2:9143	Sol1:6594 Sol2:6594	27.87925189 27.87925189	72.12074811 72.12074811	522 0
21.	Sol1:233*37 Sol2:223*41	Sol1:8621 Sol2:9143	Sol1:6072 Sol2:6594	29.56733558 27.87925189	70.43266442 72.12074811	0 522
22.	Sol1:233*37 Sol2:233*37	Sol1:8621 Sol2:8621	Sol1:6072 Sol2:6072	29.56733558 29.56733558	70.43266442 70.43266442	0 0
23.	Sol1:233*37 Sol2:233*37	Sol1:8621 Sol2:8621	Sol1:6072 Sol2:6072	29.56733558 29.56733558	70.43266442 70.43266442	3319 0
24.	Sol1:241*22 Sol2:233*37	Sol1:5302 Sol2:8621	Sol1:2753 Sol2:6072	48.07619766 29.56733558	51.92380234 70.43266442	0 3319
25.	Sol1:241*22 Sol2:241*22	Sol1:5302 Sol2:5302	Sol1:2753 Sol2:2753	48.07619766 48.07619766	51.92380234 51.92380234	0 0
26.	Sol1:241*22 Sol2:241*22	Sol1:5302 Sol2:5302	Sol1:2753 Sol2:2753	48.07619766 48.07619766	51.92380234 51.92380234	0 0
27.	Sol1:241*22 Sol2:241*22	Sol1:5302 Sol2:5302	Sol1:2753 Sol2:2753	48.07619766 48.07619766	51.92380234 51.92380234	0 0

The Simulation time on Core2Duo @3.00 GHZ machine with 1 GB RAM using MATLAB 7 environment is 1.669393 seconds. In these test cases all the Modules are hard modules i.e. the area is fixed and the aspect ratio is also fixed. Rotation of the modules is also not permitted. It is clearly shown here that only those solutions are considered which results in reduction of placement cost. So, we can say that it is a heuristic based approach of finding a global solution for placement of modules in a given area. Table 5 depicts the simulation results iteration wise by showing the net reduction in terms of dead space. In order to prevent from falling into local minima we have used some variants of mutation functions.

Mut1: In this the position of any block is exchanged by position of another block.

Mut2: In this the relation between any two blocks are reversed.

Mut3: A finite string of blocks with operators is taken and is reversed.

These variants are tried and one which gives maximum reduction is considered and remaining is discarded. It helps us in situations when we are not improving by simple inversion operation in mutation operator.

It was also observed that by increasing the number of initial solutions we can further reduce the packaging cost as shown below

1) Initial solution count = 2, Final Placement cost = 5302
2) Initial solution count = 3, Final Placement cost = 4536
3) Initial solution count = 6, Final Placement cost = 4012
4) Initial solution count = 10, Final Placement cost = 3920

The simulation results for Test Case II with 25 circuit blocks are hereby shown in figures from 18-28.

The Current analysis performed over various test cases demonstrates that it is not possible for a solution to evolve in each iteration .The randomness introduced will help in preventing from falling in local minima. But as it progresses further improvements will be resulted due to exchange of operators and operands between parents and evolution of good ones will be resulted only.

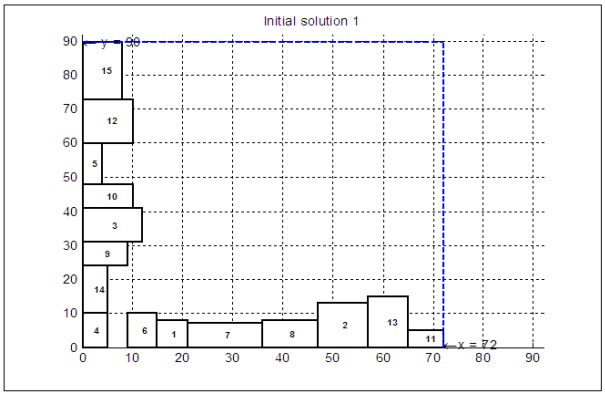


Figure 5. Initial Population: Solution 1:72*90

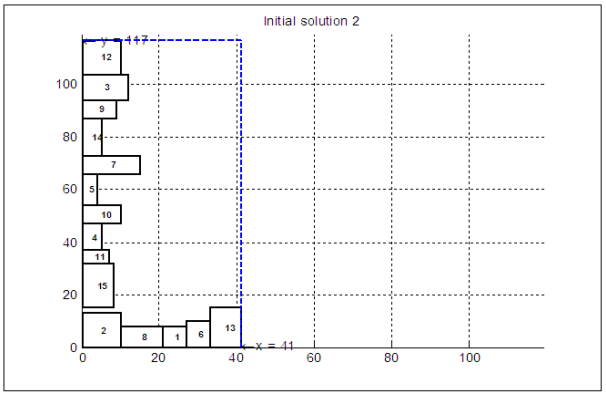


Figure 6. Initial Population: Solution 2:41*117

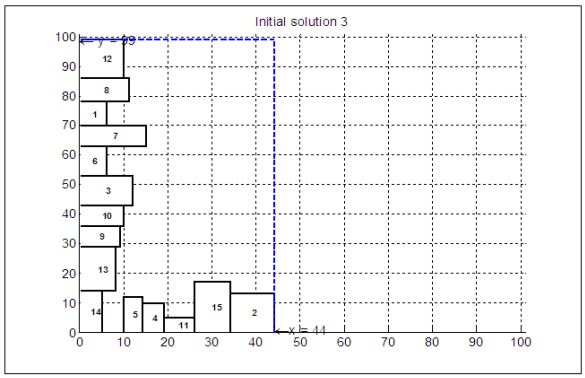


Figure 7. Initial Population: Solution 3:44*9

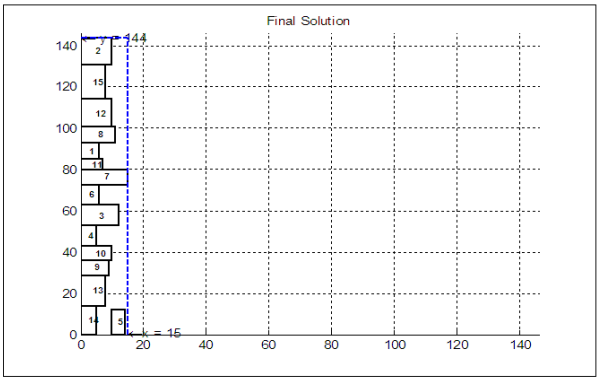


Figure 8. Final Solution:15*1

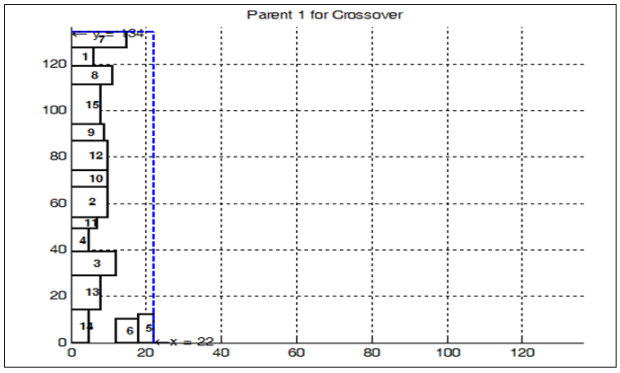


Figure 9. Parent 1 for Crossover :22*134

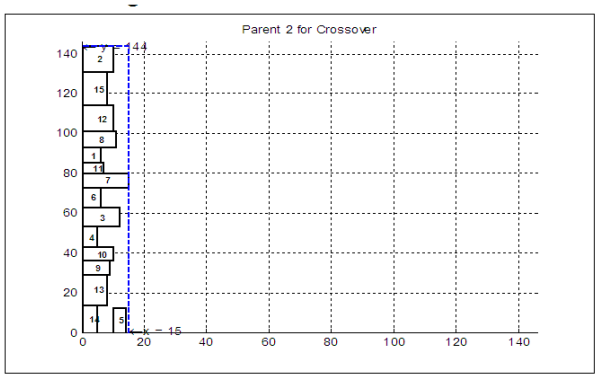


Figure 10. Parent 2 for Crossover:15*144

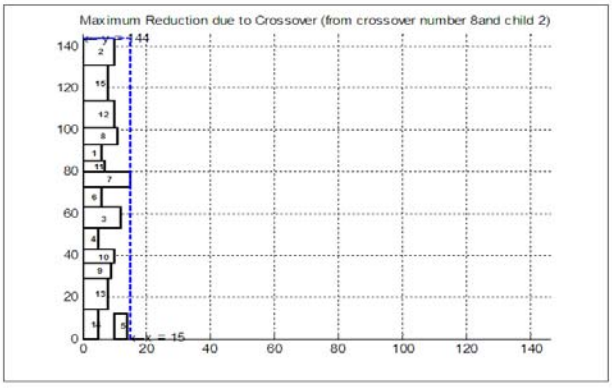


Figure 11. Result of Crossover Operator:15*144

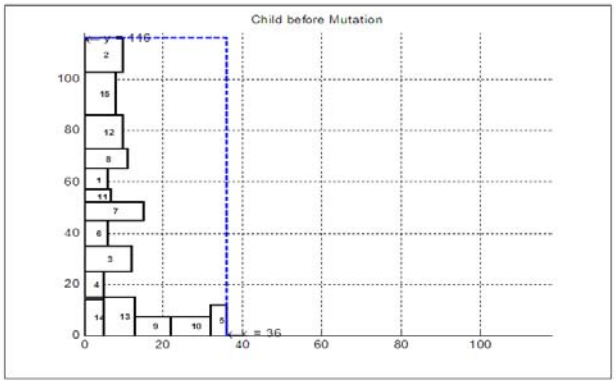


Figure 12. Child considered for mutation:36*116

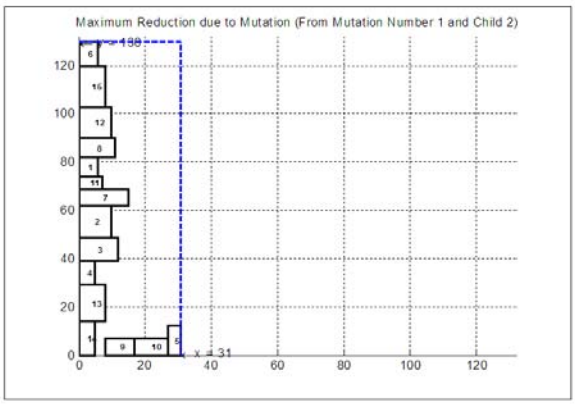


Figure 13. Child obtained after Mutation:31*130

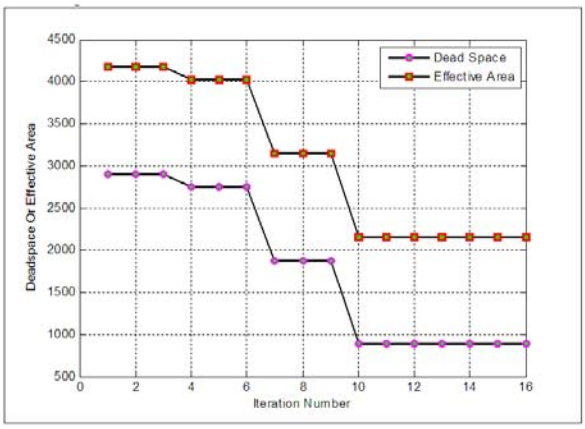


Figure 14. Net variation of Bounding Rectangle and Dead space with respect to iteration

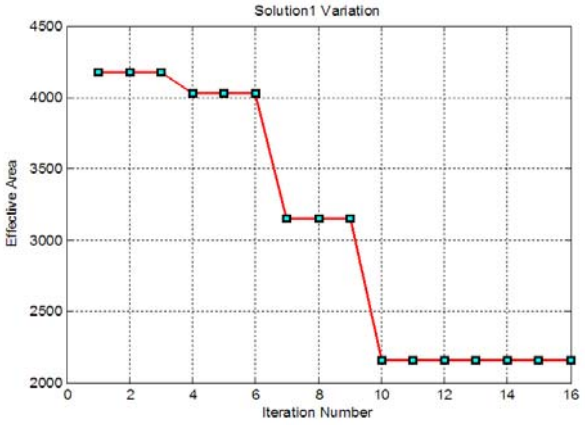


Figure 15. Variation of Solution 1 with respect to iterations

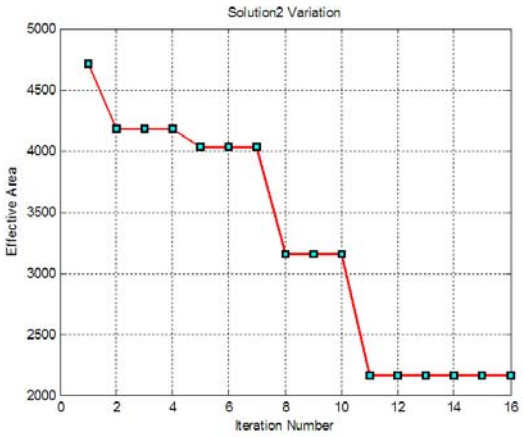


Figure 16. Variation of Solution 2 with respect to iterations

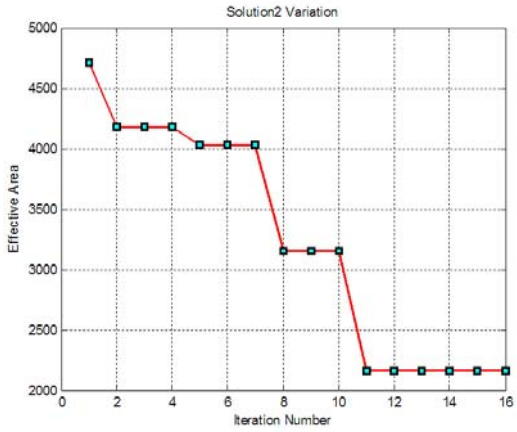


Figure 17. Variation of Solution 3 with respect to iterations

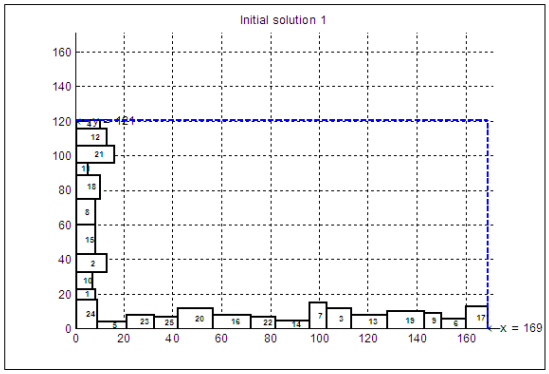


Figure 18. Initial Population: Solution 1:169*121

ISSN 1927-6974 E-ISSN 1927-6982

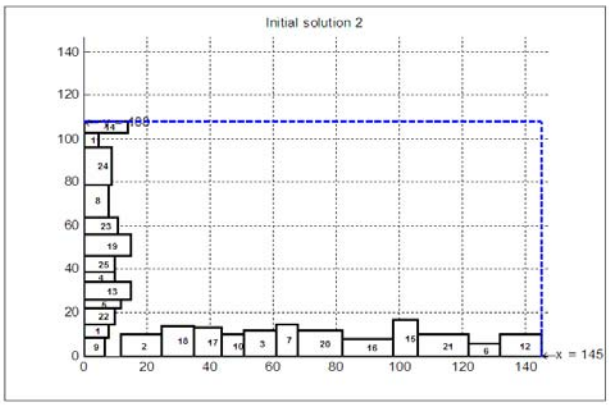


Figure 19. Initial Population: Solution 2:145*108

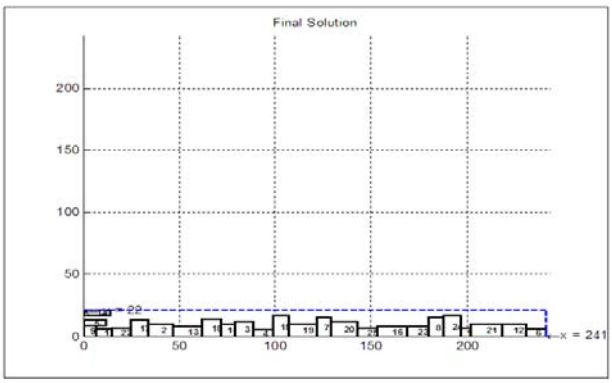


Figure 20. Final Solution:241*22

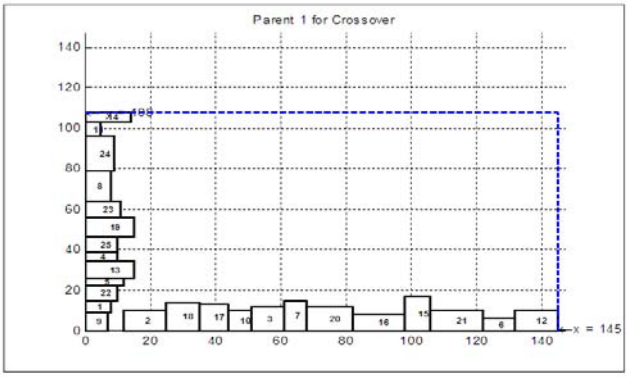


Figure 21. Parent 1 for Crossover:145*108

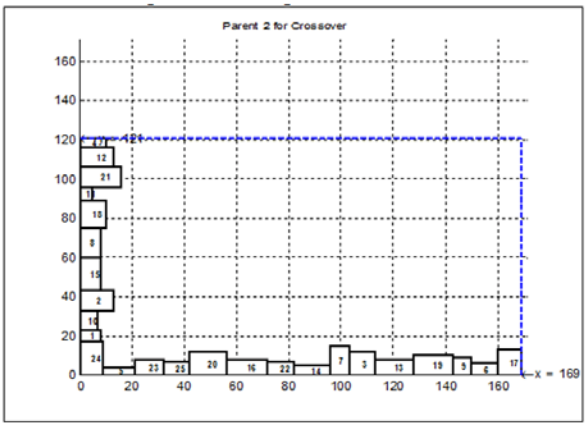


Figure 22. Parent 2 for Crossover:139*123

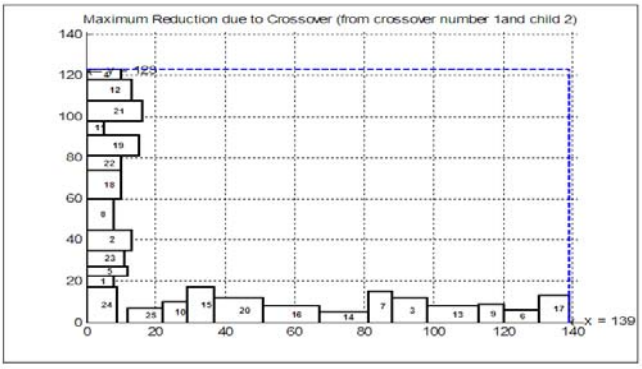


Figure 23. Result of Crossover operator:139*123

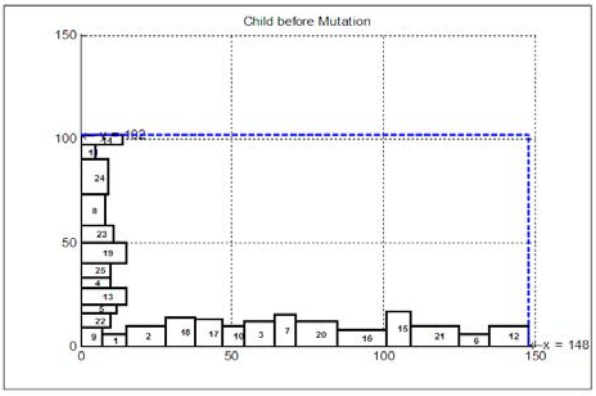


Figure 24. Child considered for mutation:148*102

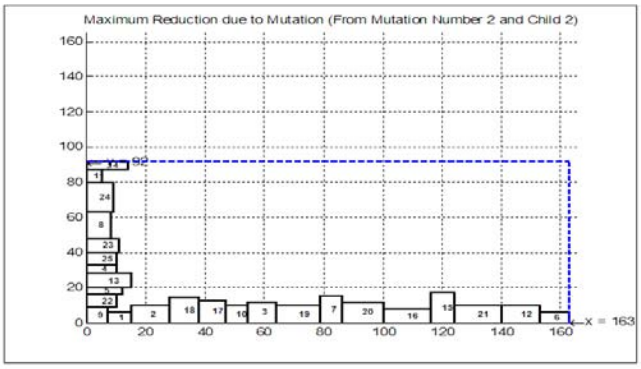


Figure 25. Child obtained after mutation:163*92

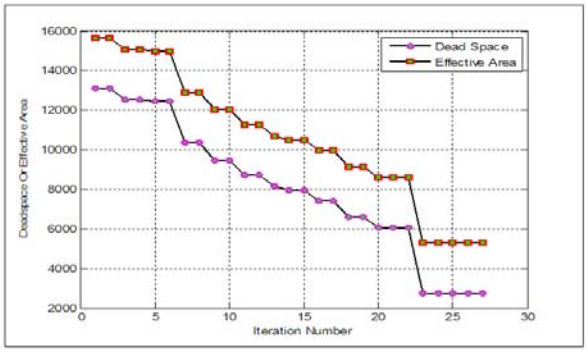


Figure 26. Net variation of Bounding Rectangle and Dead space with respect to iteration

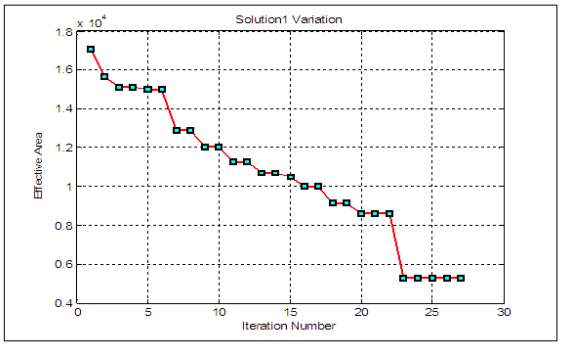


Figure 27. Variation of Solution 1 with respect to iterations

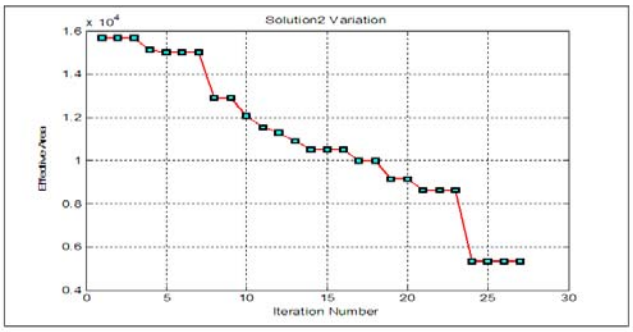


Figure 28. Variation of Solution 2 with respect to iterations

6 Conclusion

Module placement is an important step in VLSI design as it is responsible for the assignment of the circuit's cell to their locations on the chip. The main reason behind the popularity of Genetic Algorithms is that they work on a set of solutions rather than single solutions. This feature along with unique operators like crossover and mutation allows exploring and combining diverse placement ideas in a variety of situations. The probabilistic approach allows a speedy directed search of the state space towards desirable solutions. We have used modified approach of simple GA in which we have encoded our solutions using postfix expressions and instead of just randomly moving or exchanging the information between population elements, our algorithm searches for possibilities in each step and chooses the best one.

Our conclusion is that our algorithm gives optimal solutions for smaller number of modules. However, we found that it will be better to incorporate the rotation feature in the blocks as it may lead to better adjustment to the available space. By allowing the iterations to proceed unchecked we may find a very good solution but it will be resource intensive operations so we limited it if no improvement is taking place after optimal number of iterations.

References

[1] Dhiraj Sangwan, Seema Verma and Rajesh Kumar. "An Optimized Approach for Module Placement in VLSI Circuits using Genetic Algorithms," International Journal of Computational Intelligence and Information Security (IJCIIS). 2011 January; 2(1): 4-17.

[2] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake and Yoji Kajitani. “VLSI Module Placement Based on Rectangle-Packing by the Sequence Pair”, ",IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems. 1996; 15(12): 1518-1524

[3] H.Chan, P.Mazumder and K.Shahookar. “Macro-Cell and Module Placement by Genetic Adaptive Search with Bitmap-Represented Chromosome,” Integrated, The VLSI Journal. 1991 November; 12(1): 49-77. http://dx.doi.org/10.1016/0167-9260(91)90042-J

[4] James P.cohoon, William D.Paris. "Genetic Placement ", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 1987 November; 956-964.

[5] Pinaki Mazumder, Elizabeth Rudnick. Genetic Algorithms: for VLSI Design, Layout & Test Automation,Pearson Education Co.,1/e, 1999.

[6] S. Sathiamoorthy. “LaySeq: A New Representation for Non-Slicing Floorplans”. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.6579

[7] Christine L. Valenzuela and Pearl Y.Wang. “A Genetic Algorithm for VLSI Floorplanning”, Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, Springer Verlag, 671-680

[8] C.Valenzuela and P.Wang. “VLSI Placement and area optimization using a genetic algorithm to breed normalized postfix expressions,” IEEE Transactions on Evolutionary Computation. 2002 August; 6(4): 390-401. http://dx.doi.org/10.1109/TEVC.2002.802872

[9] K. Shahookar , P. Mazumder. GASP: a Genetic Algorithm for Standard cell Placement, Proceedings of the conference on European design automation.1990 March; 12-15. Glasgow, Scotland. http://dx.doi.org/10.1109/EDAC.1990.136728

[10] H.Esbensen. “A Genetic Algorithm for Macro Cell Placement, ” Proc of the European Design Automation Conference. 1992 September; 52-57. http://dx.doi.org/10.1109/EURDAC.1992.246265

[11] Sipakoulis, G.C.; Karafyllidis, I.; Thanailakis, A. "Genetic Partitioning and Placement for VLSI Circuits", Proceedings of 6th IEEE International Conference on Electronics, Circuits and Systems. 1999; 1647-1650

[12] Yosuke Kimura, Kenichi Ida. “Floorplan design problem using improved genetic algorithm", Journal ofArtificial Life and Robotics. 2004 November; 8(2): 123-126. http://dx.doi.org/10.1007/s10015-004-0298-4

[13] Suphachai Sutanthavibul, Eugene Shragowitz, J. Ben Rosen," An Analytical Approach to Floor plan Design and Optimization.",Proc of 27th ACM IEEE Design Automation Conference.1990; 187-192

[14] Gary Kok-HooYeap,and MajidSarrafzadeh."A Unified Approach to Floorplan Sizing and Enumeration",IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems. 1993; 12(12): 1858-1867

[15] Kurdahi, F.J.,Parker, A.C. “Techniques for Area Estimation of VLSI Layouts",IEEE Transactions onComputer-Aided Design of Integrated Circuits and Systems. 1989 January; 8(1): 81-92

[16] Schnecke, V.; Vornberger, O. “Genetic Design of VLSI Layouts”, Proc of First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications. 1995. GALESIA.1995; 430-435.

[17] Youssef, H.; Sait, S.M.; Nassar, K.; Benten, M.S.T.” Performance driven standard-cell placement using the genetic algorithm” Fifth Great Lakes Symposium on VLSI. 1995; 124-127.

[18] Maolin Tang, Xin Yao. “A Memetic Algorithm for VLSI Floor planning,” IEEE Trans., Systems, Man and Cybernetics. 2007 February; 37(1): 62-69. http://dx.doi.org/10.1109/TSMCB.2006.883268